Q

MOONS'

Q User’s Guide

Version Ac

Contents

Introduction

This training guide is provided as help in getting started with MOONS' Q programmer compatible drives("Q drive" for short hereafter in the document). Now, MOONS' Q drives available include MSST5-Q, MSST10-Q, MSST5-I, MSST10-I, MSSTAC6-Q, MSSTAC6-I and STM23Q. For more information regarding your *specific* Q drive, including wiring and mounting, please consult the Hardware or User's Manual for your drive.

All Q drives can be programmed with the Q Programmer software. This software allows you to create sophisticated, single-axis motion control programs.

Q drive features include:

- Simultaneous stored program execution and serial host command functionality
- Data registers
- Multi-tasking
- Math functions
- Multi-drop RS-485 for communication with more than one drive
- Robust communication protocol
- Analog Inputs
- Velocity Mode Jogging
- Password Protection for Stored Programs

Q Programmer software features include:

- Intuitive interface
- One-click buttons for most functions
- Active data register monitor
- Continuous drive status with Automatic Polling
- Host command line for sending commands to drive
- Easy-to-use editor for creating programs

Getting Started with your Q drive

It is strongly recommended that all users familiarize themselves with the Overview, Commands, and Data Registers sections of this guide. These sections provide essential information for working with Q drives. The Overview section describes the basic archictecture of Q drives, with some diagrams that show how the drives function internally. The Commands section describes the two different types of commands available in Q drives (immediate and buffered), and how they are used. The Data Registers section lists all of the data registers available in Q drives, and how to access them. After that, because this User’s Guide covers information for using Q drives in different types of applications, you may benefit from reading only the sections that pertain to your specific application or project.

Stand-Alone (Stored Program) or Host Control

You will want to determine whether your Q drive will be running a stored program or not. If so, you will need to familiarize yourself with the Q Programmer software, which is described in the Programs section of this guide. Q Programmer is provided free with all Q drives, and provides a simple way of creating and editing stored programs for Q drives. If your application does not require a stored program, and therefore your Q drive will be receiving all commands from a host controller, you will need to choose a serial connection - RS232, two-wire RS485, or four-wire RS485 - and familiarize yourself with both the Host Serial Connections and Host Serial Communication Protocols sections of this guide.

In many applications it will be common for a Q drive to do both of the above: that is run a stored program and receive commands from a host controller in the same application. If this describes your application, you will still need to determine the best serial connection and communication protocol for your host controller. The Q Programmer software running on a Windows-based PC only communicates with a Q drive via RS232, and a compatible programming cable is provided with each Q drive. However, you are still free to choose from the three available serial connections - RS232, two-wire RS485, or four-wire RS485 - for communicating between the Q drive and your host controller.

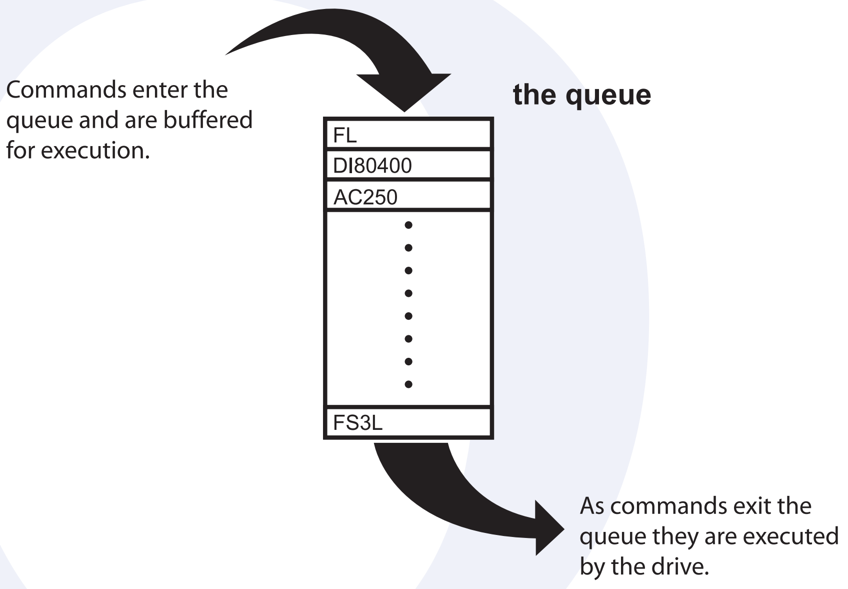
Single or Multi-axis Host Control

If you are planning to send commands to your Q drives from a host controller, you may also opt to connect more than one drive to your host controller’s serial port. 2-wire and 4-wire RS-485 allow you to connect more than one drive to a host’s serial port. Furthermore, 2-wire and 4-wire RS-485 allow you to use longer communications cables - up to 1000 feet - than with RS-232, which we suggest limiting to 50 feet. If you plan to connect more than one drive to your host controller’s serial port, you will find more information in the Host Serial Connections section of this guide.

Overview

What does the letter "Q" mean?

The "-Q" designation of our Q drives comes from the word *queue*. This is because these drives operate by executing commands that are fed into a queue, or buffer, located in the controller section of the drive. By definition, a queue is a list or sequence of items in which items enter at one end of the queue and exit at the other. In the case of our drives, the items in the queue are commands, and when a command reaches the end of the queue it is executed by the controller. Here is a basic diagram of how this works.


Commands enter the queue and are buffered for execution.
the queue
FL
DI80400
AC250
FS3L
As commands exit the queue they are executed by the drive.

The queue is physically located in a volatile area of the controller's memory. We stress volatile because at power down any commands that are left in the queue and have not been executed will be lost.

How are commands placed into the queue?

Commands enter the queue either via a drive's serial port (from a host) or from the drive's own non-volatile memory. Initially, we always send commands to a Q drive via its serial port. Most often this will be done using the Q Programmer software running on a Windows-based PC and connected to the drive via a PC COM port (RS-232). However, commands can also be sent by other programmable devices (hosts), like HMIs, microprocessors, or other types of machine controls, via either the drive's RS-232 or RS-485 port. These other devices only need to follow the communication protocols required by the Q.

Once serial communication has been established, we can do one of two things with the commands we send. We can simply place commands into the queue for immediate or buffered execution, or we can route commands via the queue into the drive's non-volatile memory storage. We refer to the latter as programming, and one of the benefits of programming is that programs stored in non-volatile memory are not lost when a drive is powered down.

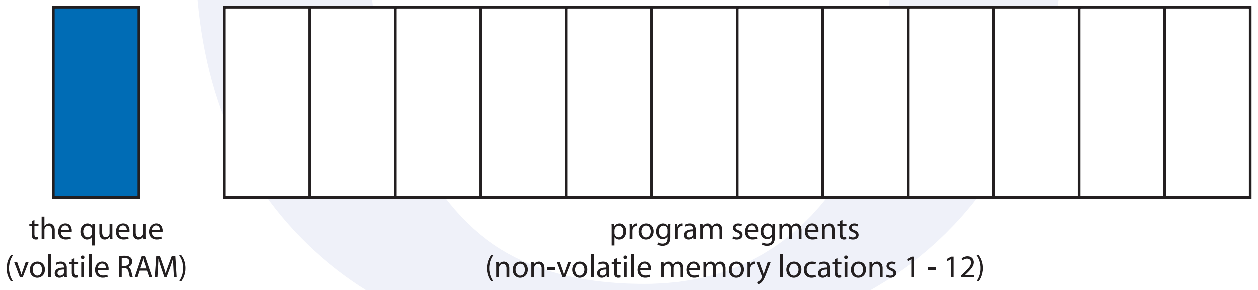
Programs in Q Drives

To understand a little more how we program a Q drive - that is, place commands in a drive's non-volatile memory - lets take a look at some terminology.

A command is comprised of two or three ascii characters (only letters are used for the commands themselves). Some commands also have parameters (letters, numbers, and other ASCII characters are used for parameters) that follow the inital two (or three) letters. There are two basic types of commands: immediate and buffered. An immediate command executes immediately, regardless of what is already ahead of it in the queue. A buffered command works by waiting its turn in the queue behind other buffered commands that may already be stored in the queue.

NOTE: The queue is large enough to hold 62 commands at any given time. We create programs for Q drives by sequencing commands together into the queue.

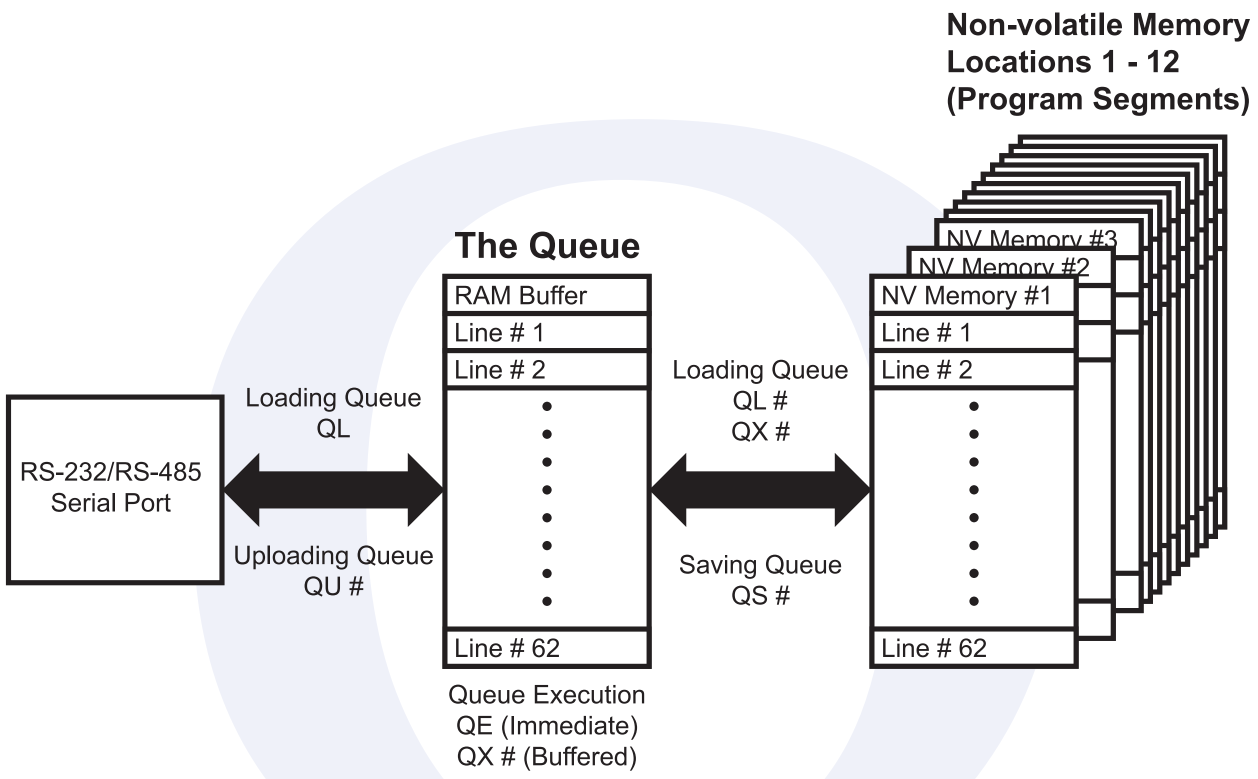
Now, while 62 of our powerful commands can create a lot of functionality, we realize that many applications will require more than this number. We address this difference when we access a drive's non-volatile memory. This non-volatile memory used for storing commands is physically much larger than the queue, and we break it up into locations that are the same size as the queue.



There are 12 non-volatile memory locations located in each drive, and in each location we can store a sequence of commands. A sequence of commands stored in a specific non-volatile memory location is referred to as a program segment. Program segments are 62 lines long, the exact same size as the queue. Any of the 12 program segments can be immediately loaded from its corresponding non-volatile memory location directly into the queue. When we program a Q drive we use specific commands to load segments into the queue. For example, we may start our program with segment 1, and at the end of segment 1 place a command that loads segment 2 into the queue. The segment loading process takes a mere 125 microseconds, so the jump from segment to segment is basically seamless.

Now you can see that with 12 program segments linked together and 62 lines in each segment, you can create programs in a Q drive with up to 744 commands. This is a significant number of commands, and we think it's enough to make very powerful, flexible, and funcitional programs. So the key to programming a Q drive then lies in using the 12 available program segments together, to create a program with the desired functionality.

The next diagram shows a little more detail of how the queue is accessed from both the serial port of the drive (RS-232/RS-485) and the drive's own non-volatile memory.



In the above diagram, the serial port of a drive is used to access the queue using Load (QL) and Upload (QU) commands. The Load command allows a sequence of commands to be placed in the queue by a host device (PC, HMI, microprocessor, etc.), and the Upload command allows that same device to upload the contents of the queue.

Furthermore, the Load command can also be used to load a program segment from a non-volatile memory location into the queue. The Save (QS) command allows the contents of the queue to be saved as a program segment in a non-volatile memory location.

Part of the flexibility of these drives is that program segments can be loaded and executed in any order. There are only a couple of important rules. Segment 1 is always loaded first when a drive is set to run its stored program at power up (stand-alone). You can think of Segment 1 as a kind of auto-exectuion segment. Also, when using the input interrupt function, On-Input (OI), the program will always default to Segment 10. Therefore your command sequence for what to do after an interrupt should always be placed in Segment 10. Aside from these two rules, you are free to jump from segment to segment in any order.

Commands

As stated in the Overview section of this guide, commands are executed by a Q drive once they've been fed into the queue. Commands are fed into the queue from two sources, a drive's serial port via a host and a drive's non-volatile memory. In all applications you will start using a Q drive by sending commands to it serially. Then, if your application calls for your Q drive(s) to run a stored program, you will save commands into the drive's non-volatile memory. The rest of this section describes the types of commands available in Q drives.

Command Structure

All commands in Q drives are made up of three possible parts: the command itself (also referred to as the command code), and two parameters. Some commands never use parameters, some always use parameters, and others change their function slightly depending on whether or not they are followed by a parameter. In general, the structure is like this:

Command(Parameter 1)(Parameter 2)

where "command" is usually two letters (some commands are made up of three letters), and the "parameters" 1 and 2 are made up of ascii characters - letters, numbers, and other ascii characters. Here are a few examples.

FL

Feed to Length command - no parameters - executes an incremental move

SO1L

Set Output command - Parameter 1 = "1" , Parameter 2 = "L" - sets physical drive output 1 low

RL91000

Register Load command - Parameter 1 = "9", Parameter 2 = "1000" - loads the User-Defined data register 9 with the value 1000.

For a complete reference of commands, their syntax, parameters, and usage, please consult the Q Command Reference, which is a separate document.

Buffered and Immediate Commands

There are two types of commands: buffered and immediate. Buffered commands are stacked in the queue one on top of another. They are executed in the order they enter the queue. If you send two buffered commands into the queue in succession, the second command will not be executed until the first command finishes. (See section on multi-tasking for an exception to this rule).

Immediate commands are executed right away, regardless of what's happening in the queue, and therefore can be executed in parallel with a buffered command.

All commands are either buffered or immediate.

Only buffered commands can be used to create a stored program. The reason is simple. In a stored program, commands are sequenced together because of timing with external events in the application. An immediate command does not obey the timing of a sequence.

When using a host controller device to send commands to your Q drive you can use both immediate and buffered commands. The choice over one or the other will often be very simple because certain drive functions are naturally commanded with immediate or buffered type commands. However, some drive functions can be commanded with either an immediate or equivalent buffered command, and in these cases you will have to choose which command type is best in your application. To illustrate this some more, here are some sister commands in their immediate and buffered forms.

Alarm Reset: AR (Immediate) or AX (Buffered)
Stop Move: ST (Immediate) or SM (Buffered)
Queue Kill: SK (Immediate) or QK (Buffered)
Register Load: RL (Immediate) or RX (Buffered)
Queue Load & Execute: Combination of QL, QE (Immediate) or QX (Buffered)

Buffered Command Categories

What follows is a listing and the descriptions of the 7 categories of buffered commands. These categories are the same that you will find when editing programs in the Q Programmer.

Motion
Servo
Configuration
I/O
Communication
Q Program
Register

Motion Commands

Motion commands relate to position and velocity control of the motor shaft: incremental/relative moves, absolute moves, conditional moves, jogging, stopping motion, etc.; as well as commands that affect the parameters of these moves: acceleration, deceleration, jog speeds, maximum acceleration, velocity limits, etc.

Servo Commands

Servo commands are for tuning and servo status. This category includes commands for adjusting the PID loop, filters, and velocity loop terms. It also includes commands for enabling and disabling the servo.

Configuration Commands

Configuration commands are for setting peak and continuous current levels, defining mulit-drop address, setting position fault and limit, encoder resolution, and more.

I/O Commands

I/O commands affect the digital inputs, analog inputs, and digital outputs of the drive. For digital inputs these include wait for input, defining limits, filtering inputs, defining alarm input, defining servo enable input, and more. For the analog inputs there are commands for filtering the input, setting a threshold, offsetting and zeroing the input. For the digital outputs there are commands for the alarm, brake and motion outputs, as well as setting general purpose outputs.

Communication Commands

This category contains three commands for setting the communication protocol, adjusting the bit rate, and setting the transmit delay.

Q Program Commands

There are three subcategories in this category: Q, Miscellaneous, and Wait. The Q subcategory consists of commands used for loading, uploading, and saving commands to and from the queue, as well as branching, looping, and calling within a program. Miscellaneous commands include those for multi-tasking, interrupts, password protection, and others. The wait subcategory has four commands: wait for input, wait for move to finish, wait time, wait delay register.

Register Commands

Register commands are for doing math on register data and moving data into and between registers. This category includes commands for counting, decrementing, incrementing, adding, subtracting, multiplying, dividing, AND, OR, comparing registers, moving data, reading, writing, loading, and more.

Buffered Command Listing

Here is a listing of the buffered commands available with the Q drives, in alphabetical order by category.

Motion Commands

AC	Acceleration
AM	Maximum Acceleration
CJ	Commence Jogging
DC	Change Distance
DE	Deceleration
DI	Distance
EG	Electronic Gearing
FC	Feed and Change Velocity
FD	Feed to Double Sensor

FE	Follow Encoder
FL	Feed to Length
FM	Feed and Mask Sensor
FO	Feed and Set Output
FP	Feed to Position
FS	Feed to Sensor
FY	Feed to Sensor with Distance Limit
HW	Hand Wheel
JA	Jog Accel/Decel
JD	Jog Disable
JE	Jog Enable
JM	Jog Mode
JS	Jog Speed
SH	Seek Home
SM	Stop Move
SP	Set Absolute Position
VC	Change Velocity
VE	Velocity
VM	Maximum Velocity

Servo Commands

AX	Alarm Reset
EP	Encoder Position
KC	Servo Filter
KD	Derivative Gain
KE	Derivative Filter
KF	Velocity Feedforward Gain
KI	Integral Gain
KK	Acceleration Feedforward Gain
KP	Proportional Gain
KV	Velocity Feedback Gain
KW	Velocity Feedback Filter
MD	Motor Disable
ME	Motor Enable
VI	Velocity Mode Integral Gain
VP	Velocity Mode Proportional Gain

Configuration Commands

CC	Change Continuous Current
CM	Control Mode
CP	Change Peak Current
DA	Define Address
ER	Encoder Resolution
PC	Power Up Current
PF	Position Fault

PL	Positioning Limit
PM	Power Up Operating Mode
PP	Power Up Peak Current
PT	Positioning Time
SA	Save All NV Parameters

I/O Commands

AF	Analog Filter
AI	Alarm Reset Input
AO	Alarm Output
AT	Analog Threshold
AV	Analog Offset
AZ	Analog Zero
BD	Brake Release Delay
BE	Brake Engage Delay
BO	Brake Ouput
DL	Define Limits
FI	Filter Input
MO	Motion Output
OI	On Input
SI	Servo Enable Input
SO	Set Output
SOY	Set Output Extended
TI	Test Input
WI	Wait for Input

Communication Commands

BR	Bit Rate
PR	Protocol
TD	Transmit Delay

Q Program Commands

MT	Multi-tasking
OF	On Fault
OI	On Input
PS	Pause
PW	Password
SS	Send String
WI	Wait Input
WM	Wait Move
WT	Wait Time
WD	Wait Delay Register
QC	Queue Call
QG	Queue Goto
QJ	Queue Conditional Jump

QK	Queue Kill
QR	Queue Repeat
QU	Queue Upload
QX	Queue Load & Execute

Register Commands

CR	Compare Register
DR	Data Register for Compare
RC	Register Counter
RD	Register Decrement
RI	Register Increment
R+	Register Add
R-	Register Subtract
R*	Register Multiply
R/	Register Divide
R&	Register AND
R|	Register OR
RM	Register Move
RR	Register Read
RW	Register Write
RX	Register Load
SR	Set Register Pointer
SV	Set Register Value
TS	Time Stamp Read

Immediate Command Listing

What follows is a listing of immediate commands available in Q drives. These commands are always going to be used by a host device (never in a stored program), so they focus primarily on status and parameter requests and other immediate-type functions.

AC	Alarm Code
AR	Alarm Reset
BS	Buffer Status
CE	Comm Error
CS	Change Speed
CT	Continue
GC	Current Command
IA	Immediate Analog value
IC	Immediate Commanded Current
ID	Immediate Distance request
IE	Immediate Encoder request
IF	Immediate Format
IH	Immediate High output
IL	Immediate Low output
IP	Immediate Position request
IO	Immediate Output status request
IQ	Immediate Current value
IS	Input Status request
ISX	Expanded Input Status request
IT	Immediate Temperature
IV	Immediate Velocity
IU	Immediate DC Bus Voltage
IX	Immediate Position Error
MN	Model Number
QE	Queue Execute
QL	Queue Load
QS	Queue Save
RE	Re-start or Reset
RL	Register Load
RS	Request Status
RU	Register Upload
RV	Revision Level request
SC	Request Status in Hexadecimal
SJ	Stop Jogging
SK	Stop & Kill buffer (queue)
ST	Stop

Data Registers

Many of the commands described in the preceding section function by transfering data to a drive for later use. The data values sent by these commands are stored in data registers inside the drive. Values remain in registers until new commands change them or until power is removed from the drive.

NOTE: Some registers are automatically saved in non-volatile memory and therefore not lost at power-down of a drive. Examples of these are the registers affected by the PC (Power on Current) and PM (Power on Mode) commands. Also, you can choose any data registers you’d like to store in non-volatile memory by placing them in non-volatile data register storage locations using the RW command.

An example of a command that transfers data to a drive for later use is the DI (Distance) command. This command places a value in the “D” register, and this value is used for many other commands like FL (Feed to Length), FS (Feed to Sensor), CJ (Commence Jogging), and others. In effect, the function of the DI command is to change the contents of the “D” register.

There is another command that can change the contents of the “D” register, and it is the Register Load command (RL is the immediate version, RX is the buffered version). This command can also place a value in the “D” register. In fact, the Register Load command can place values in any “writeable” registers, and you will find that it is an extremely useful command.

There are many data registers in each Q drive. The following sub-sections detail the three types of data registers available.

Read-Only Registers

Read-only registers are predetermined registers that contain information about drive parameters, settings, and states. These include registers for the commanded current in the servo amplifer, the motor's encoder position, analog input levels, drive temperature, internal bus voltage, and more. These registers cannot be changed by the user, but they can be monitored by the user. All read-only registers are represented by a lower-case letter. Here is a listing of available read-only registers.

a	Analog Command
b	Queue Line
c	Current Command
d	Relative Distance
e	Encoder Position
f	Alarm Code
g	Sensor Position
h	Condition Code
i	Driver Inputs
j	Analog Input 1
k	Analog Input 2
l	Comm Error
m	Control Mode
n	Velocity State
o	Position State
p	Segment Number
q	Actual Current
r	Average Clamp Watts
s	Status Code
t	Drive Temperature
u	Bus Voltage
v	Velocity
w	Velocity Feedforward
x	Position Error
y	Move Token
z	Phase Error

Read/Write Registers

Read/write registers are predetermined registers that contain drive and move parameters that can be set by the user. These parameters include acceleration rate, velocity, move distance, continuous current, peak current, and more. Read/write registers are represented by capital letters. Here is a listing.

A	Acceleration
B	Deceleration
C	Change Distance
D	Distance
E	Encoder Position
F	Other Flags
G	Current Command
H	Reserved
I	Input Counter
J	Velocity Command
K	Jog Acceleration
L	Jog Deceleration
M	Maximum Velocity
N	Continuous Current
O	Peak Current
P	Position Command
Q	Reserved
R	Steps per Revolution
S	Step Position
T	Total Count
U	Change Velocity
V	Velocity
W	Time Stamp
X	Analog Position Gain
Y	Analog Threshold
Z	Analog Offset

User-Defined Registers

User-defined registers are read/write registers that are not predetermined. These registers are available for you to create more flexible and powerful programs. These registers are represented by single-digit numbers and other ASCII characters. Here is a list of the available user-defined registers.

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
:	(colon)
;	(semi-colon)
<	(less than)
=	(equals)
>	(greater than)
?	(question mark)
(	(left parenthesis)
\	(backslash)
)	(righ parenthesis)
^	(“Shift+6”)
_	(underscore)
‘	(apostrophe)

Non-Volatile Data Register Storage

In addition to the three types of data registers mentioned above, there are 125 non-volatile memory locations that can be used to store data register values. Each of the 125 non-volatile memory locations can store the contents of one data register. None of the data registers are associated with a particular non-volatile memory location, so it is up to you to remember which memory location you are using when writing a data register to non-volatile memory. See the following sections on Writing and Reading data registers for more information.

Accessing Data Registers

Loading

Accessing data registers is done by Loading data into a register, and Uploading data from a register. Loading a data register can be done from a host command line or from a line in a program. To load a register from a host command line use the RL (Register Load) command. This command can be executed at any time, even while a drive is running a program. The RL command is an immediate command. To load a register within a program we use the RX command, which is a buffered version of Register Load.

Uploading

Uploading data registers can only be done from a host command line, not within a program. The command to upload a register is RU (Register Upload). It is an immediate command, and therefore can be executed while a program is running. The upload command has an extra feature that is designed to work with host command systems. A second parameter can instruct the command to send up to 10 data registers in sequence back to the host command device. This is great when an array of information is required at one time.

Writing

Writing a data register allows the user to store data register values in non-volatile memory. To write a data register we use the RW (Register Write) command. There are 125 storage locations for data registers in NV memory. Note that the user must keep track of where data registers are stored because the NV memory locations are not associated with any specific data register.

Reading

Reading a data register allows the user to move data previously saved in NV memory into a data regster. To read a data register we use the RR (Register Read) command. Reading is typically done in the midst of a program.

Manipulating Data Registers

Moving

Data register values can be moved from one register to another. This is done with the RM (Register Move) command. When executing an RM command, the contents of the originating data register are retained. Contents of read-only registers can be moved into read/write registers and user-defined registers. However, as implied by its label, no register values can be moved into read-only registers. Attempting to do so will have no effect and no error code is generated.

Incrementing/Decrementing

Read/write and user-defined registers can be incremented and decrmented by “1”. Two commands are used for these functions: the RI (Register Increment) and RD (Register Decrement) command. NOTE: Incrementing past the range of a data register will cause the value to wrap around.

Counting

A special data register, the "I" register (Register Counter), is designated for counting input transitions and input state times of a selected digital input. The "I" register is a read/write register that can be used with all other register functions including math and conditional testing.

The RC (Register Count) command is used to assign digital inputs to register counting. There are four different input states that can be chosen and that have different effects on input counting. When using the "high" or "low" level states the counter acts as a "timer" with a resolution of 125 microseconds. Edge type states like "falling" or '"rising" are used for input counting. (See details of the RC command in the Q Command Reference).

Math & Logic

Math and logic functions can be performed on data registers. Math is limited to integer values. Some of the math functions are also limited to 16-bit values. When doing math only one operation can be done per instruction. Math and logic results are stored in the Accumulator register, "0". This register is part of the user-defined register set. Math functions include Add, Subtract, Multiply and Divide. Logic functions include Logical AND and Logical OR.

Conditional Testing

When constructing complex programs it is usually necessary to do some conditional processing to affect program flow. Two commands are available for evaluting a data register for conditional processing, the TR (Test Register) and CR (Compare Register) commands. The TR command wil compare the "First" value of a given data register against a "Second" immediate value. The CR command compares the "First" value of a given data register against the "Second" value of another data register. When using the TR and CR commands an internal "Condition" register is set with the result. The result can be:

"True"	the "First" value is either positive or negative
"False"	the "First" value is not a value (it's zero)
"Positive"	the "First" value is "positive"
"Zero"	the "First" value equals "0"
"Greater Than"	the "First" value is more positive than the "Second" value
"Less Than"	the "First" value is more negative than the "Second" value
"Equal"	the "First" and "Second" values are equal
"Unequal"	the "First" and "Second' values are not equal

NOTE: The QJ (Queue Jump) command is designed to use the "Condition" codes above for jumping.

Host Serial Connections

Introduction

When communicating to a Q drive you will always be using one of the following serial connections: RS-232, 2-wire RS-485, or 4-wire RS-485. Out of the box we suggest starting with RS-232 along with the programming cable and software that was supplied with your Q drive, so that you may be communicating to and familiarizing yourself with the Q drive as quickly as possible. All software from MOONS’ communicate to a Q drive via the supplied RS-232 programming cable. These software include:

Q Programmer ------------	create and edit stored programs, emulate a host
SCL Setup Utility ---------	basic host terminal for host emulation

If your project calls for a Q drive (or drives) running stored programs, you will use the supplied RS-232 programming cable along with Quick Tuner and Q Programmer to setup, configure, and program your drive(s). If your project calls for your drive(s) only running stored programs, you can read up on the RS-232 sub-section in this section and not read any more about the other serial connections. However, if your application calls for a serial host controller (PC, PLC, HMI, or other serial device that can act as a host) being able to communicate to the drive(s), you will need to choose one of the three available serial connections.

Available Host Serial Connections: RS-232, 2-wire RS-485, 4-wire RS-485

When choosing the best serial connection for your project, the choice may be made for you based on the host controller you plan to use. For example, some devices only communicate via 2-wire RS-485. If you are not restricted by your host controller, here are two guidelines for choosing the best connection.

Single or mutli-axis

If your project calls for communicating to only drive you can consider any of the three options. If your project calls for communicating to more than one drive you should use 2-wire or 4-wire RS-485.

Long communication cables

In many applications, the limitation of 50 feet on RS-232 will be sufficient. In applications where the distance between drive and host controller will be more than 50 feet (up to 1000 feet), you will need to choose 2-wire or 4-wire RS-485.

A Quick Summary of 2-wire and 4-wire RS-485 connections

The 2-wire and 4-wire RS-485 protocols that Q drives utilize are based on industry standard RS-485 and RS-422 protocols. Strictly defined, RS-485 is a 2-wire interface that allows multi-node connections limited to half-duplex serial communications. Up to

32 nodes that both transmit and receive can be connected to one network. On the other hand, RS-422 in the strictest definition is a 4-wire point-to-point connection that allows full-duplex serial communications when connected to a single node. RS-422 has one node that is the driver or transmitter and up to 10 nodes that are receivers. RS-422 was not designed for a true multi-node network.

The Q drives are designed to work in a multi-node environment, and so they use both the standard 2-wire RS-485 connection, and a modified RS-422 (4-wire) connection that has been termed "4-wire RS-485". This is because unlike the standard RS-422, which is desinged for single-node connections, the 4-wire RS-485 used by Q drives allows multiple nodes.

NOTE: In general we recommend using half-duplex communications with Q drives. Even though the 4-wire RS-485 network can support full-duplex, their is now the capability to have multiple nodes and therefore data collisions might occur. For this reason we recommend limiting communications to half-duplex, even with the 4-wire RS-485 connections.

Throughout this User's Guide we refer to the three possible serial connections as "RS-232", "2-wire RS-485", and "4-wire RS-485".

Connecting to your Q drive's serial port(s)

Each Q drive comes with two physical connectors for connecting to a PC or other serial host controller device. The first connector is an RJ11 connector (same as a 4-wire phone jack) that is used strictly for RS-232 communications. The second connector is a removable 5-position terminal block for use with 2-wire and 4-wire RS-485 connections.

The following paragraphs illustrate the various connections.

Connecting to a PC using RS-232

Each Q drive comes with a programming cable for use with the drive's RS-232 port. This cable is made up of two parts, a 7 foot 4-wire cable (looks just like a 7 foot telephone cord), and an RJ11 to 9-pin DSUB adapter. This adapter allows you to connect to the COM port (serial port) of your PC. Here are the general directions for connecting your Q drive to your computer.

♦ Locate your computer within 6 feet of the Q drive.

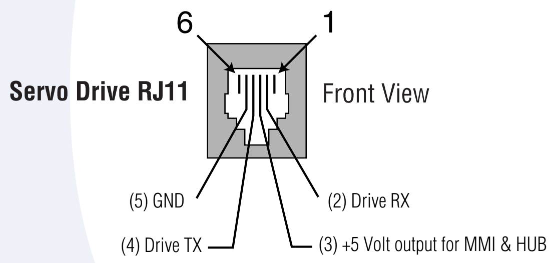
♦ Plug the 9-pin end of the adapter supplied with your Q drive to the COM1 serial port of your PC. Secure the adapter with adapter's two screws. If the COM1 port on your PC is already used by something else, you may use the COM2 port of your PC. On some PCs, COM2 will have a 25-pin connector rather than a 9-pin. If this is the case with your PC, and you must use COM2, you will have to purchase a 25 to 9 pin serial adapter at your local computer store.

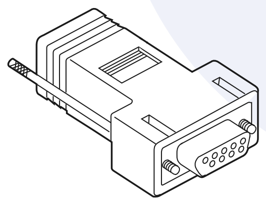
NOTE: If you are using a laptop computer that does not have any COM ports, only

USB ports, you will have to use a USB to Serial adapter. There are a variety on the market, and some work better than others. But in general, once you’ve installed the USB to Serial adapter, your PC will assign the adapter a COM port number. Remember this number when you go to use your MOON’ software. Also, if you are having troubles with your adapter, contact MOONS’ for help with recommended adapters.

♦ Now take the 7 foot cable and plug one end into the adapter you just attached to your PCs COM port, and plug the other end into the RS-232 (RJ11) jack on the Q drive. If you need to locate your drive farther from the PC, you can replace the 7 foot cable with any 4-wire telephone cord. Do not exceed 50 feet.

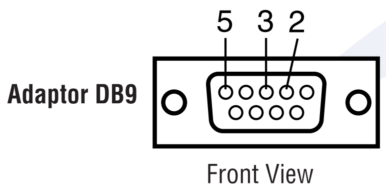
WARNING: Never connect a Q drive or other MOONS’ drive to a telephone circuit. It uses the same connectors and cords as telphones and modems, but the voltages are not compatible.

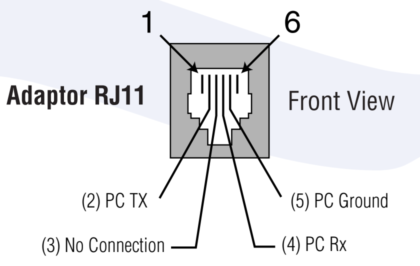




DB9 to RJ11 Adaptor Pin assignments

DB9 female	Signal Name	RJ11	Signal Name
2	RX	4	TX
3	TX	2	RX
5	GND	5	GND





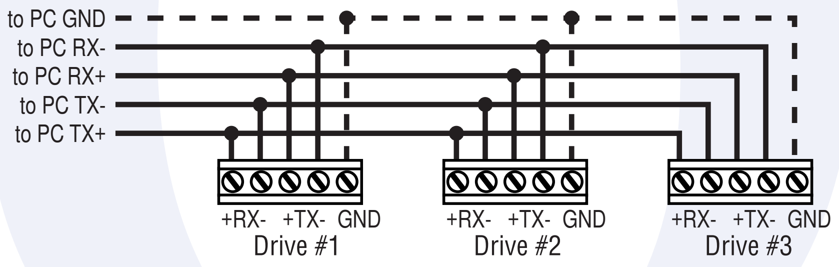
PC DB9 to
Servo Drive RJ11 Adaptor

NOTE: Pins 1 & 6 are not connected

Connecting to a host using 4-wire RS-485

Our 4-wire RS-485 implementation is a multi-drop network with separate transmit and receive wires. One pair of wires connects the host's TX+ and TX- signals to each drive's RX+ and RX- terminals. Another pair connects the RX+ and RX- signals of the host to the TX+ and TX- terminals of each drive. A common ground terminal is provided on each drive and can be used to keep all drives at the same ground potential. This terminal connects internally to a drive's ground connection, so if all the drives on the 4-wire network are powered form the same supply it is not necessary to connect the logic grounds. You should still connect one drive's GND terminal to the host's signal ground. Before wiring the entire system you'll need to connect each drive individually to the host so that a unique address can be assigned to each drive. (See following sub-section "Before you connect the servo drive to your system"). Proceed as follows, using the figure below.

1. Connect the drive TX+ to the host/PC RX+.
2. Connect the drive TX- to the host/PC RX-.
3. Connect the drive RX+ to the host/PC TX+.
4. Connect the drive RX- to the host/PC TX-.
5. Connect GND to the host/PC signal ground.



Getting and Connecting an RS-485 4-wire adapter to your PC.

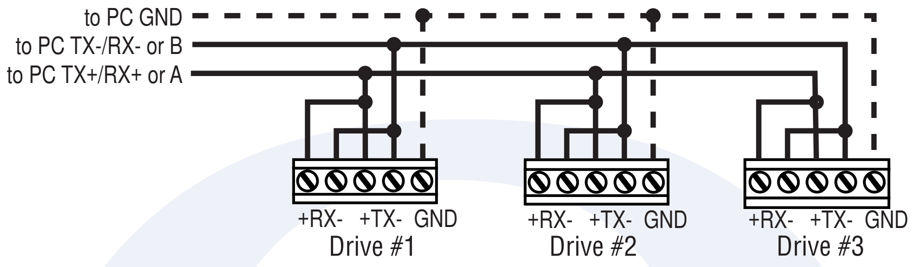
If you are using your computer to communicate to the Q drive(s) and therefore need an RS-485 adapter, model 117701 from Jameco Electronics (800-831-4242) works well. This adaptor is for a 25-pin serial port. If you are like most people and have a 9-pin serial port on your PC, you will also need to purchase Jameco cable 31721. Connect as follows:

Adaptor Terminal	Drive Terminal
1	RX+
2	RX-
3	TX-
4	TX+

Set the switches on the Jameco adaptor for DCE and TxON, RxON. Don't forget to plug in the DC power adapter that comes with the unit.

Connecting to a host using 2-wire RS-485

Our 2-wire RS-485 implementation is a multi-drop network with one pair of wires that is used for both transmit and receive. To make this type of connection you will first need to jumper the TX+ terminal of a drive to it's own RX+ terminal, and then do the same with the TX- and RX- terminals. To then connect a drive to the host/PC, you will need to connect the TX+/RX+ terminals of the drive to the host/PC's TX+/RX+ terminal, and then the TX-/RX- terminals of the drive to the host/PC's TX-/RX- terminal. Here is a diagram.



Getting and Connecting an RS-485 2-wire adpater to your PC.

If you are using your computer to communicate to the drive(s) and therefore need an RS-485 adaptor, model 485-25E from Integrity Instruments (800-450-2001) works well. It comes with everything you need. Connect as follows:

Adaptor Terminal	Drive Terminals
A	TX+/RX+
B	TX-/RX-

Before you connect the servo drive to your system

If you plan to implement a 2-wire or 4-wire RS-485 network of Q drives, you will first need to address each drive individually. An easy way to do this is prior to hooking your drives up with one of the RS-485 implementaions, using the RS-232 cable that came with each Q drive, and the SCL Setup Utility. If you've already connected your Q drive using one of the RS-485 implementations, completing this sub-section will allow you to test your connections.

First connect your PC and Q drive. (See preceding sub-sections on connecting to a PC or host for help with this). Then launch the SCL Setup Utility on your PC. If you don't have the SCL Setup Utility installed, you can get it either from the CD-ROM that came with your Q drive or from MOONS' website, www.moons.com.cn.

Once the SCL Setup Utility is launched, select the proper COM port of your PC, and then apply power to the Q drive. Press the Caps Lock key on your keyboard (because the drives only accept commands in uppercase). Type RV then press Enter. If the drive has power and is properly wired, it will respond with "RV=x", where x is the firmware version of your drive. This confirms that communication has been established. If you don't see the "RV=x" response, check your wiring and follow the above procedures again.

Next, you must choose an address for each drive. Any of the "low ascii" characters (many of which appear above the number keys on a PC keyboard) are acceptable:

! “ # $ % & ‘ ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < > ? @

To find out which address is already in a Q drive, type DA then press Enter. The drive will respond with “DA=x”, where x is the address that was last stored. To change the address, type “DAy”, where y is the new address character, then press Enter.

To test the new address, type “yRV” where y is the address you’ve just assigned to the drive, and then press Enter. For example, if you set the addresss to % and want to test the address, type “%RV” then press Enter. The drive should respond with “%RV=x” where x is the firmware version of the drive.

Once each drive in your network has been given a unique address, you can proceed to wiring the whole system together.

Host Serial Communication Protocols

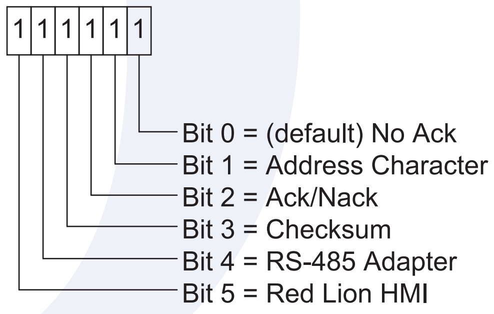
Because of the intense nature of serial communications required in Q drive applications, you are allowed to adjust a drive's serial communications protocol to best fit your application. This adjusting of a drive's serial communications protocol is done using the PR command.

Typically the PR command is used one time when configuring a drive and saved as part of the startup parameters. However, it can be changed at any time to dynamically alter the serial communications.

The PR command works by sending the decimal equivalent of a 5-bit binary "word". Each bit in the word represents a different setting of the serial communication protocol. These settings are additive, meaning when you set a bit to "1", or turn it on, you are adding the functionality of that setting to the serial protocol. Think of this 5-bit word as a bank of 5 dip switches. You can turn each dip switch on or off, and in doing so add or subtract a particular setting from the overall protocol.

The PR command in detail

The diagram to the right shows the assignments of each of the 5 bits in the protocol word. Remember that when you use the PR command the parameter that you send along with the command code (PR) is the decimal equivalent of this binary word. Below are the details of each of the bits and the settings they are assigned to.



Bit 0 - (Default) "No Acknowledge" (AKA "Standard SCL")

When commands that do request returned data are received by the drive, no other response is sent from the drive

Send data Examples:

Tx	DI8000	Global set distance to 8000
Rx	nothing	
Tx	1DI8000	Drive with address "1" set distance to 8000
Rx	nothing	

Request data Examples:

Tx	DI	Global distance request
Rx	DI=8000	Drive responds with distance
Tx	1DI	Distance request from drive at address "1"
Rx	1DI=8000	Drive "1" responds with distance

Bit 1 - "Always send Address Character"

With this option set (Bit 1=1) a drive's address character will always be returned with any requested data.

Send data Examples:

Tx	VE50	Global set velocity to 50 rps
Rx	nothing	
Tx	1VE50	Drive with address "1" set velocity to 50 rps
Rx	nothing	

Request data Examples:

Tx	VE	Global velocity request
Rx	1VE=50	Drive responds with velocity and address "1"
Tx	1VE	Velocity request from drive at address "1"
Rx	1VE=50	Drive responds with velocity and address "1"

Bit 2 - "Send Acknowledge Character (Ack/Nack)"

This option causes the drive to acknowledge every transmission from a host, whether the command is requesting data or not. If a host requests data (for example a DI command with no parameter), the response is considered the acknowledgement. However, if the host sends commands that do not request data from the drive, the drive will still respond with one of the following characters:

"%" - The "percent" character is a Normal Acknowledge (Ack) character that means the drive accepted the command and executed it.

"*" - The "asterisk" character is an Exception Acknowledge (Ack) character that means the drive accepted the command and buffered it into the queue. Depending on the state of the queue, command execution can occur at any time after the acknowledge.

"?" - The "question mark" character is a Negative Acknowledge (Nack) character that means an error occured while the drive was receiving the command. A second character may follow the question mark, which provides an error code describing the type of error. Here is the list of error codes:

Negative Acknowledge Codes

1 = Command timed out
2 = Parameter is too long
3 = Too few parameters
4 = Too many parameters
5 = Parameter out of range
6 = Command buffer (queue) full
7 = Cannot process command
8 = Program running

Acknowledge characters are always sent out of the RS-232 port. When operating on a 2-wire or 4-wire RS-485 network, the Acknowledge characters are sent out under the following conditions:

1. An Acknowledge character is sent when the received command has an Address character at the beginning.

2. An Acknowledge character is NOT sent when Global commands (commands without Adresses) that do not request data from the drive are used.

3. Global commands that request data will cause data to be returned from a drive. This can cause data collisions if there are more than one drive on a network. Use Addresses with commands to avoid this problem.

NOTE: When possible avoid using Acknowledge characters (%, *, ?) as drive Addresses to prevent confusion.

Good command Example:

Tx	DI8000	Global set distance to 8000
Rx	%	Normal Ack is returned (RS-232 only)
Tx	1DI8000	Drive "1" set distance to 8000
Rx	1%	Drive "1" sends normal Ack

Bad command Example:

Tx	VE200	Global set velocity to 200 rps
Rx	?5	Negative Ack (Nack) is returned (RS-232 only)
Tx	1VE200	Drive "1" set velocity to 200 rps
Rx	1?5	Drive "1" responds with Negative Ack (Nack)

Buffered command Example:

Tx	AC10	Global set Acceleration to 10 rps/s
Rx	*	Exception Ack is returned (RS-232 only)
Tx	1AC10	Drive "1" set Acceleration to 10 rps/s
Rx	1*	Drive "1" responds with Exception Ack

Bit 3 - "Use 8-bit Checksum"

Not implemented at this time. Call factory for schedule.

Bit 4 - "Special RS-485 adapter mode"

Allows using the drive as an RS-485 adapter by letting the host communicate on an RS-485 network through a drive's RS-232 port. When the host sends commands with a "~" (tilde) at the beginning of the command to the drive's RS-232 port, the command is echoed out of both the drive's RS-232 and RS-485 ports. Drives connected to the RS-485 network will receive the same command with the "~" stripped off.

Without the Bit 4 option (Bit 4=0), a Q drive will normally echo any addressed command out of the RS-232 port only, whether the command was received from the drive's RS-232 or RS-485 port. What the Bit 4 option (Bit 4=1) does, is force the drive to echo commands out the RS-485 port as well, allowing a host that is connected to a drive through it's RS-232 port, to communicate to an RS-485 network of drives.

NOTE: When both Bits 4 and 2 are set (Bit 4=1, Bit 2=1), the host will receive back both the echoed packet and the Acknowledge packet. For example: two drives are connected in an RS-485 network, and they both have PR command Bits 4 and 2 set. The first drive, which is also connected to the host via its RS-232 port, is addressed "1", and the second drive is addressed "2". Here is what you will see:

Send data Example:

Tx	~2DI8000	Drive "2" set distance to 8000
Rx	2DI8000	Echoed packet from drive "1" (RS-232 and RS-485)
Rx	2%	Drive "2" responds with Normal Ack

Request data Example:

Tx	~2DI	Drive "2" request distance
Rx	2DI	Echoed packet from drive "1" (RS-232 and RS-485)
Rx	2DI8000	Drive "2" responds with distance

Bit 5 - "Red Lion 3 character Data Register number"

Each data register in a Q drive is normally accessed using its single letter, number, or other ascii character. (See the Data Register section for more character assignments). With Bit 5 set (Bit 5=1), each of the data registers is instead accessed with a 3 character number: 000 to 074. (See Q Command Reference for registers' equivalent 3 character numbers). The Bit 5 option implements this specific usage for the RL (Register Load) and RU (Register Upload) commands. This option was added to make the Q drives compatible with HMI's from Red Lion Controls.

NOTE: When data is returned from a drive with (whether Bit 5 is set or not set), the data register is always represented by its single character designation.

RL Command Example:

Tx	RL017100	Load register 017 (register A) with the value 100
Rx	nothing	
Tx	RL017	Request contents of register 017 (register A)
Rx	RLA=100	Drive sends contents of register A

RU command Example:

Tx	RU0174	Upload array of 4 registers, starting with 017 (A)
Rx	RUA=100	Contents of register A are returned
Rx	RUB=150	Contents of register B are returned

Rx	RUC=140	Contents of register C are returned
Rx	RUD=210	Contents of register D are returned

PR Command Examples

Now that you know what the bits in the PR command's 5-bit binary word mean, here are a couple examples whoing how you would set the serial communications protocol of your Q drive.

Example: Turn on Ack/Nack (Bit 2) and Red Lion function (Bit 5)

The 5-bit word for this combination is - 100100 - and it's decimal equivalent is 36. Therefore, to set your Q drive with this serial protocol, you would send "PR36" to your Q drive.

Example: Turn on RS-485 adaptor function (Bit 4)

The 5-bit word for this combination is - 010000 - and it's decimal equivalent is 16. Therefore, to set your Q drive with this serial protocol, you would send "PR16" to your Q drive.

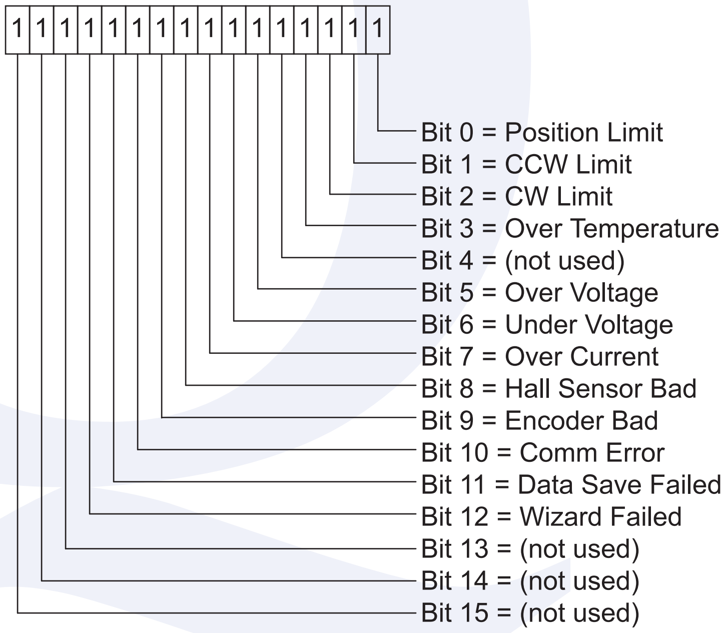
Alarm and Status Codes

One of the Q drive's diagnostic tools is its ability to send alarm and status codes back to a host. The AL (ALarm code) and SC (Status Code) commands can be used by a host to query a drive at any time. If a drive faults or sets an alarm, the AL command allows the host to find out what alarm, or alarms, has been set. Similarly, the SC command allows a host to find out what the status code of a drive is at any time during drive operation. A status code provides information as to whether the drive is running, in position, disabled, homing, and other conditions. Both alarm and status codes can be very useful when initially setting up and integrating a servo system into your machine.

The Alarm and Status codes are hexadecimal equivalents of 16 bit binary "words". Each bit in each binary word is assigned a meaning, and therefore a code word can actually show information about more than one alarm or status condition.

Alarm Code Definitions

Here is a diagram showing the meaning assigned to each of the 16 bits in the Alarm Code's binary "word". For example, when Bit 0 is logic high (= 1), it means the servo motor is in position. Similarly, if Bit 5 = 1, there is an Over Voltage condition at the drive. A drive will set any and all bits that pertain to its immediate alarm/fault condition at the moment of receiving the AL command from the host.



When a host sends the AL command, the response from the drive will actually be the Hexadecimal equivalent of this 16-bit word. This hexadecimal value is considered the Alarm Code, and the eqivalent hexadecimal value for each of the bits in the diagram is given below.

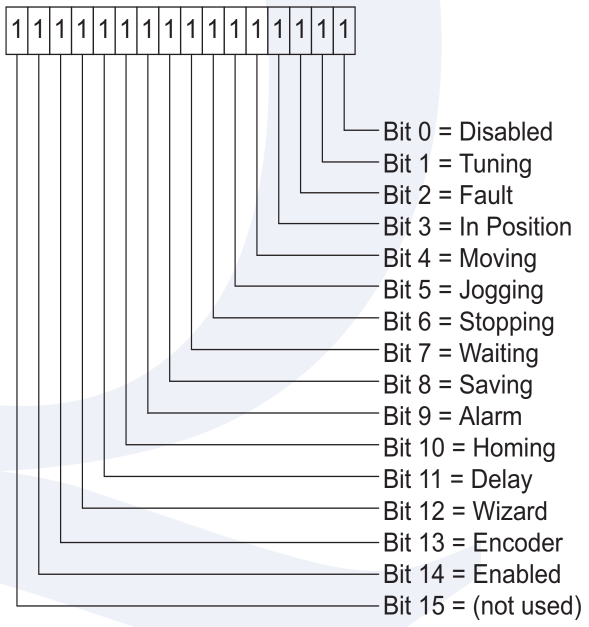
Description	Hex Value	Bit #	Comments
Position Limit	0001	0	As set by PF command
CCW Limit	0002	1	Drive input #6 activated
CW Limit	0004	2	Drive input #7 activated
Over Temperature	0008	3	Drive > 85 deg C
(not used)	0010	4	
Over Voltage	0020	5	Bus voltage > 58 volts
Under Voltage	0040	6	Bus voltage < 18 volts

Over Current	0080	7	Phase current > 20 amps
Hall Sensor Bad	0100	8	Bad Hall sensor pattern
Encoder Bad	0200	9	A or B signal not present
Comm Error	0400	10	Bad serial communications
Data Save Failed	0800	11	Unable to save data
Wizard Failed	1000	12	Timing Wizard failed
(not used)	2000	13	
(not used)	4000	14	
(not used)	8000	15	

Example: The drive has hit the CW (clockwise) limit (Bit 2), there is an under voltage condition (Bit 6), and an encoder wiring connection has been lost resulting in a bad encoder fault (Bit 9). The resulting 16-bit word is - 0000001001000100 - and the equivalent hexadecimal value is 0244. Therefore, when the host sends “AL”, the drive will respond with “AL=244”.

Status Code Definitions

Here is a diagram showing the meaning assigned to each of the 16 bits in the Status Code’s binary “word”. For example, when Bit 1 = 1, the drive is disabled. Similarly, when Bit 10 = 1, the drive is seeking the home sensor (defined by SH - Seek Home command). A drive will set any and all bits that pertain to its immediate status condition at the moment of receiving the SC command from the host.



When a host sends the SC command, the response from the drive will actually be the Hexadecimal equivalent of this 16-bit word. This hexadecimal value is considered the Status Code, and the equivalent hexadecimal value for each of the bits in the diagram is given below.

Description	Hex Value	Bit #
Disabled	0001	0
Tuning	0002	1
Fault	0004	2
In Position	0008	3
Moving	0010	4
Jogging	0020	5
Stopping	0040	6

Waiting	0080	7
Saving	0100	8
Alarm	0200	9
Homing	0400	10
Delay	0800	11
Wizard	1000	12
Encoder	2000	13
Enabled	4000	14
(not used)	8000	15

Example: The drive is enabled (Bit 14), it’s in position (Bit 3), and it’s waiting (Bit 7) for an amount of time specified by the WI command. The 16-bit word for this condition is - 0100000010001000 - and the hexadecimal equivalent is 4088. Therefore, when the host sends “SC”, the drive will respond with “SC=4088”.

A useful tool for converting alarm and status codes to binary

If you’re using a Windows-based PC as a host with your Q drive (which you’ll definitely be doing if you’re using any of the MOONS’ software supplied with your Q drive), you can use the Calculator utility that comes with Windows to conver hexadecimal values into binary values or “words”. This utility is usually found in the Accessories folder of your Programs Folder, in the Start menu. Once open, make sure the Scientific view is set by choosing it from the View menu of Calculator. This view provides some radio buttons for switching between Hex and Bin (as well as Dec and Oct).

To figure out what your Alarm or Status Code is telling you, check the Hex radio buttion and enter the hexadecimal code sent by the drive. Then check the Bin radio button and your code will automatically be converted to a binary word. Note: Calculator does not allow leading zeros in entries, so you may see less than 16 bits. That’s OK, just start counting from the right with Bit 0, and you will be able to determine the conditions set in the codes.

Programs

This section provides assistance in using the Q Programmer to create a stored program for your Q drive. If your application does not require a stored program in the drive, familiarizing yourself with the Q Programmer is still beneficial as the software provides diagnostic and troubleshooting functions that are helpful in any application.

Getting ready to program

We've taken an overview of the Q drives, and we've looked at the types of commands available. Now let's look at how we create programs in Q drives using the Q Progammer software. We've already seen that Q drives operate by executing sequences of commands in sets of up to 62 commands at a time. These sets of commands are called segments, and there are 12 segments in a program. Remember, not all segments have to be used to make a program. Some segments can be blank, but there will always be at least one segment in a program.

To start programming a Q drive, the best approach is to break your application down into stages. For example, if you've got a homing sequence, followed by some absolute moves, then a jogging routine, and finally a series of incremental moves, it might be best to devote one program segment to each stage of your application. With this in mind, you would program one segment for your homing sequence, one for your absolute moves sequence, one for your jogging routine, and then one for your incremental moves. That's only four segments so far, and you've got eight left you can use.

Keep in mind that Segment 1 is the default power-up segment, so it makes sense to put globabl parameters like accel / decel rates, interrupt settings, register values and other housekeeping commands in this segment. It is also common to turn multi-tasking on in this segment.

Next, remember to decide what interrupt handling your program has to take care of. If for example after an input interrupt you want to seek a home position, put your commands to accomodate this in Segment 10, which is the default On Input (OI) interrupt segment. Also, if you're going to add some commands for recovering from a fault, you will have to use the On Fault command (Segment 1 is a good place to put the OF command), which designates which segment the program will branch to in the case of a drive fault. In this segment should go your fault recovery commands.

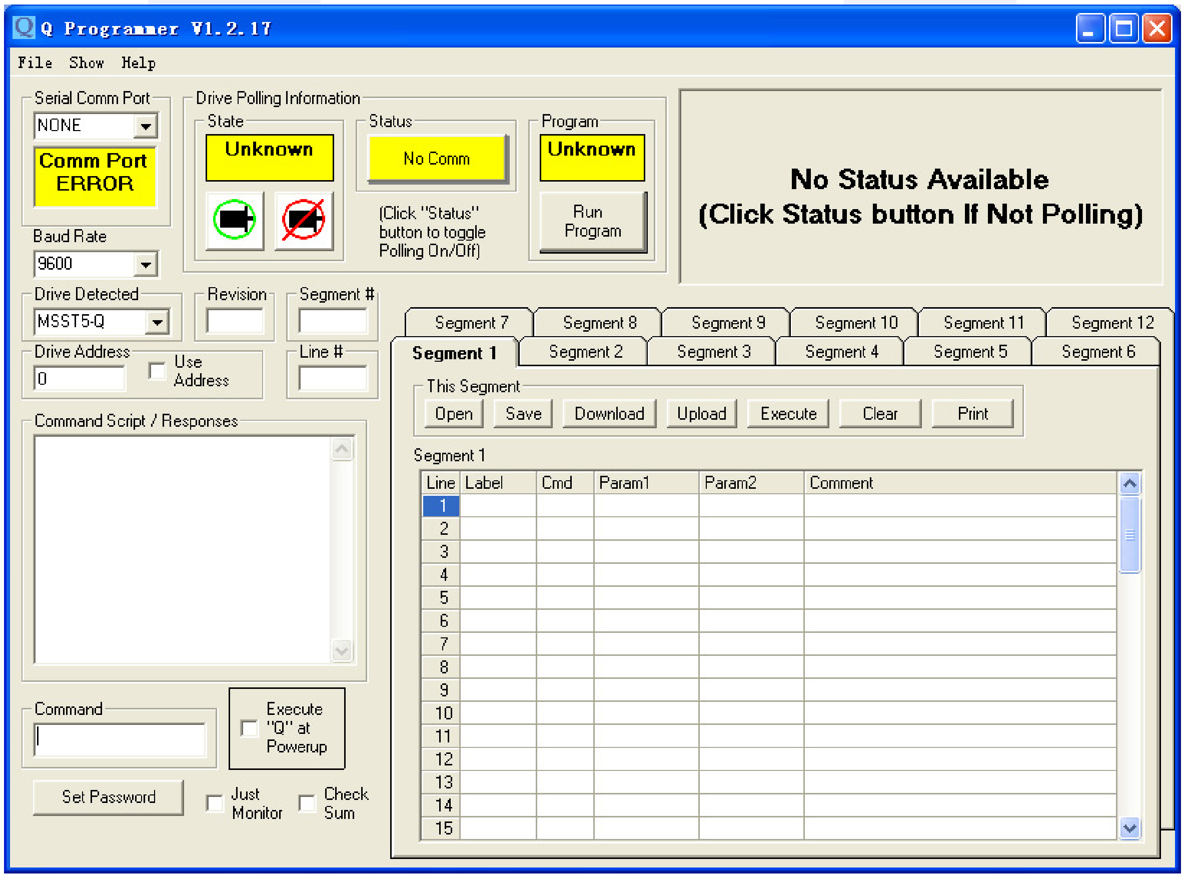
And finally, start thinking about your program and how you can break it into logical chunks or blocks of functions. Sometimes it's easiest to program the functions for one part of your application and then move on to the next part of your application, taking one application requirement at a time. After you've worked a little with the Q drives, your ability to segment an application and create a program for it will be much better.

The Q Programmer software

All Q drives can be controlled and programmed from just about any serial or host terminal device. Because the communication protocol for Q drives is based on simple serial communications, a simple terminal utility (like the SCL Setup Utility from Applied Motion) can be used to access every single function of a Q drive. However, the range of functions in a Q drive is too wide to make this type of setup really useful, and so we have created the Q Programmer software to assist users in working with Q drives as efficiently as possible. The Q Programmer software provides these major functions for the user:

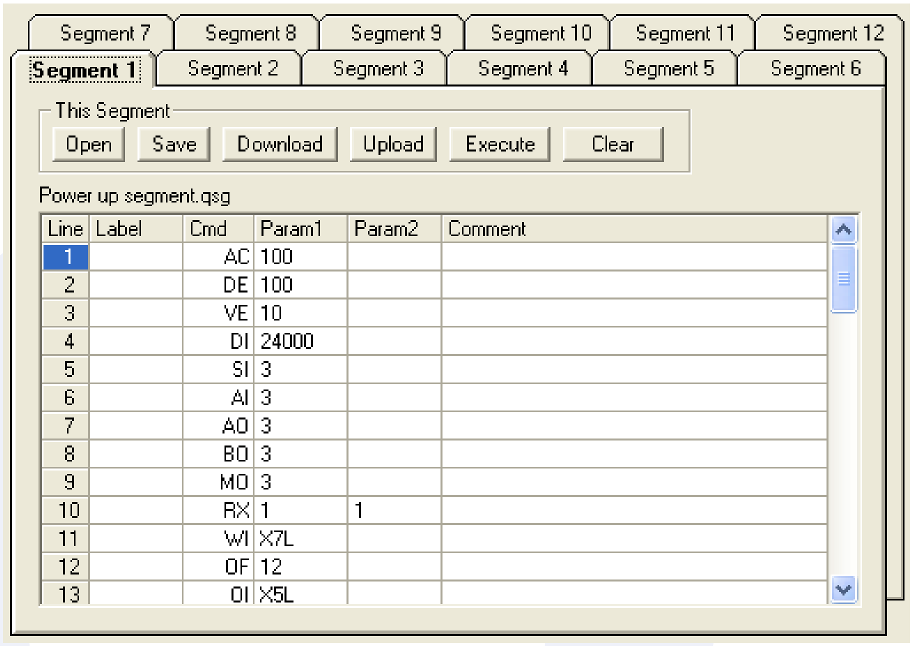
- It establishes and configures serial communication with a Q drive.
- It monitors drive status by continuously polling a Q drive.
- It provides a host command line for sending commands directly to a drive's queue.
- It helps the user create, edit and troubleshoot programs.

Here is what the main screen of the Q Programmer looks like.



Segment Editing

When programming a Q drive you will by default always be programming a particular program segment. The segment editor section of the Q Programmer has 12 tabs, one for each segment. You make a segment active for editing by clicking on its tab.



Power up segment.qsg

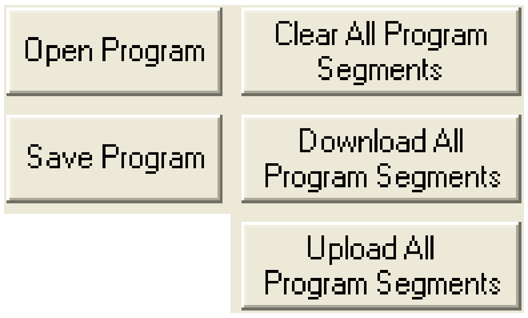
Line	Label	Cmd	Param1	Param2	Comment
1		AC	100		
2		DE	100		
3		VE	10		
4		DI	24000		
5		SI	3		
6		AI	3		
7		AO	3		
8		BO	3		
9		MO	3		
10		RX	1	1	
11		WI	X7L		
12		OF	12		
13		OI	X5L		

In the picture to the right, Segment 1 is the active segment. Taking a look at this segment tab, we see that there are several buttons underneath the tab. We have Open, Save, Download, Upload, Execute, and Clear. The Open button allows you to open a segment file (.qsg) that has been saved to disk. The Save button allows you to save the contents of the active segment to disk. The Download button sends the contents of the active tab to the Q drive. The Upload button retrieves the active segment from the Q drive. The Execute button tells the Q drive to execute the active segment (must be Downloaded first). The Clear button deletes the contents of the active segment, though only in the Q Programmer, not in the Q drive. (To clear the contents of a segment in the drive you must download a blank segment).

Each of the segment tabs also has an area for editing the sequence of commands in a segment. There are 62 lines in each segment, and for each line you can specify a Label, a Command (Cmd), Parameters (Param1, Param2) and a Comment. Labels are used for branches and jumps inside your program segment. Comments are saved with files on disk, but are not maintained in the drive. If you download a segment with comments to a drive and then upload the same segment, the comments will not be retained.

Working with Segment and Program files: .qsg and .qpr

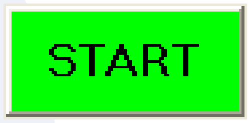
When you use the buttons available in the segment editors you are Opening, Saving, Downloading, Uploading, and Executing a particular segment. When doing these operations you are working with a .qsg file extension. In the image above you can see that the "Power up segment.qsg" file name is shown just underneath the Open and Save buttons.


Open Program
Clear All Program Segments
Save Program
Download All Program Segments
Upload All Program Segments

Looking at the main programming screen, above the segment tabs, we see there are buttons for doing the same Open/Save, Download/ Upload, and Execute operations, but with all segments together, as a program. When using these buttons we are working with a .qpr file extension. A. qpr file contains all of the individual segment files in one program file.

These buttons are useful because a program is the set of all 12 segments together, and most often we will be working with many segments in creating a program.

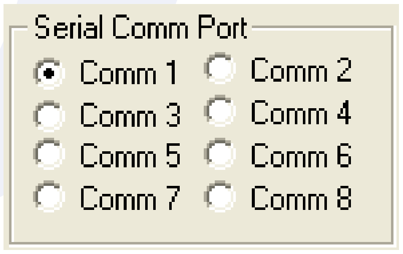
For example, if you click on the Open Program window, you will open a .qpr file from disk and all of the segment tabs will be filled with their corresponding segment files.



The START button

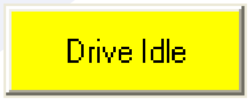
Another button that affects the entire program is the green START button. The START button initiates execution of a program that has been downloaded to a Q drive by executing Segment 1. This button simulates a power up situation when the Execute "Q" at Power Up toggle box has been selected. The START button will change its color to red and display "STOP" when a program is running.

Let's take a look at the rest of the screen to see what other functions are available.



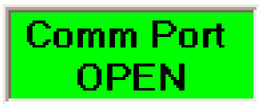
Serial Comm Port buttons

These buttons allow you to choose which comm port of your PC will be used for connecting to your Q drive. When you launch the Q Programmer the application will automatically detect an available comm port, starting with Comm Port 1. You may also manually select a comm port. Once the comm port has been established, Q Programmer will remember that comm port the next time you launch the application.



Automatic Polling

One of the Q Programmer's features is to continually poll the attached drive for status information. This feature is extremely useful in troubleshooting your drive and program. You can turn automatic polling on and off using the Drive Idle button. This button is yellow when you first launch the Q Programmer and displays the words "Drive Idle". You can activate polling by click on the Drive Idle button.



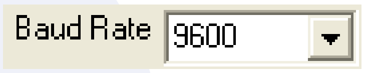
The status of polling is displayed in the Comm Port status window. This status window is located directly above the Drive Idle button. When polling is off, this window is green and displays

"Comm Port OPEN". When polling is on, this window is purple and displays "Comm Port POLLING", with the word POLLING flashing.

With automatic polling on, the Drive Idle button will...

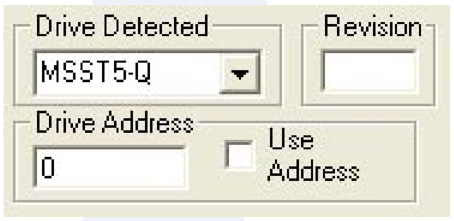
- turn green and display "Program Running" when a program is executing.
- turn red and display "Drive Faulted" if the drive has faulted and cannot recover.
- turn red and display "No Reponse" if communication with the drive is lost.
- turn yellow and display "Faulted. Program Running" if the drive has faulted but is able to recover.

If auomatic polling is off the Drive Idle button will not correlate to the actual status of the drive.



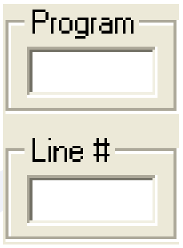
Baud Rate

This drop down menu allows you to select one of the available baud (or bit) rates. The available baud rates are 9600, 19200, 38400, 57600, and 115200 bits per second.



Drive Information and Status windows

Drive Detected will tell you which drive is connected to your computer: MSST5-Q, MSST5-I, MSSTAC6-I etc.. Revision shows the firmware level of the drive. For example, 1.52. Drive Address shows an ASCII character address that can be user defined. Changing the drive address is necessary when using the RS-485 connection of a drive in multi-drop scenarios. The Use Address toggle box is used for turning the drive address function on and off.

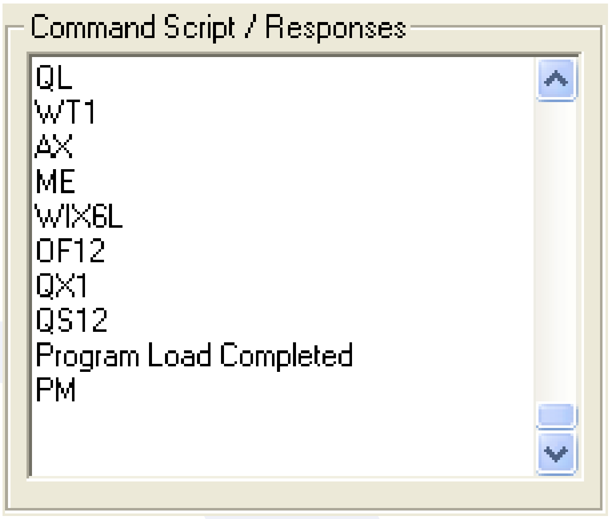


The Program window is a status indicator that shows which program segment is being executed. This number will change rapidly when a program is being executed from within the Q Programmer application. It will remain steady when there is no program executing or when a program is halted. This feature is useful for following the flow of programs created with the Q Programmer. This window only updates when polling is on.

The Line # window is just like the Program indicator, except it shows the actual line # that is being executed within a program segment. The Line # and Program indicators are very useful for monitoring a program that is executing from the Q programmer. This window is also only active when polling is turned on.

The Host Command Line

When working with a Q drive we often want to test commands and check parameters or settings either immediately or without affecting the edited segments. The Q Programmer allows us to do this by including a host command line for sending commands directly to the drive. In doing this we are not Downloading commands. Downloading is a non-volatile function: it saves program segments into non-volatile memory. When we use the host command line we are simply placing commands in the drive's queue for execution.

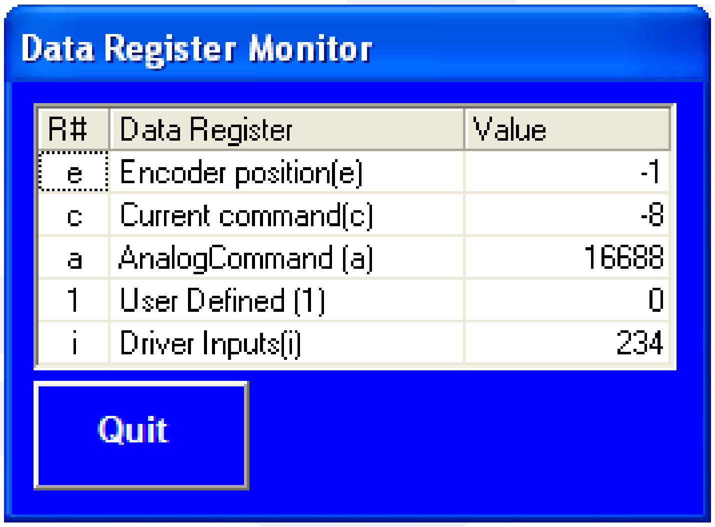

Command Script / Responses

```
QL
WT1
AX
ME
WIX6L
OF12
QX1
QS12
Program Load Completed
PM
```

The Command Script/Responses window shows a history of commands that are sent to the drive from the Q Programmer. This includes commands that are entered into the host command line, as well as commands that are sent during downloads, both program downloads and individual segment downloads.

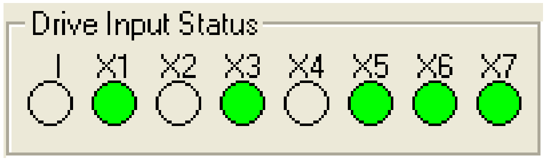
The Data Monitor

The data monitor button opens a smaller, second window in Q Programmer, which is used for monitoring up to 5 data registers. Which 5 data registers are monitored is up to the user, and for this feature to work Automatic Polling must be turned on. In the example to the right, the e, c, a, and i read-only data registers have been selected for monitoring, as well as user-defined register 1. You can select which data register you want to monitor by clicking in one of the fields in the Data Register column of the window. When you do this, a ... button will appear. This button allows you to select from all of the available data registers.


Data Register Monitor

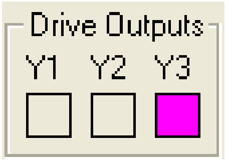
R#	Data Register	Value
e	Encoder position(e)	-1
c	Current command(c)	-8
a	AnalogCommand (a)	16688
1	User Defined (1)	0
i	Driver Inputs(i)	234

Drive I/O Status


Drive Input Status

I	X1	X2	X3	X4	X5	X6	X7
○	●	○	●	○	●	●	●

The status of digital inputs and outputs on the drive is shown in the upper right-hand corner of the Q Programmer screen. The two status indicators are only active when Automatic Polling is turned on. Drive inputs that are active are indicated with a green dot, and drive ouputs that are active are indicated with a purple square.


Drive Outputs

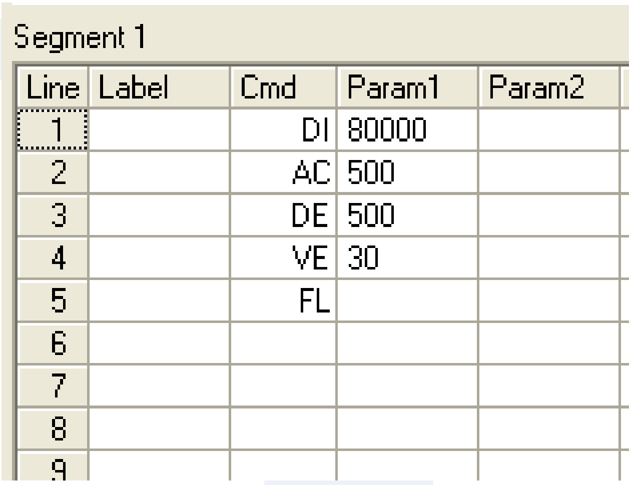
Y1	Y2	Y3
□	□	■

Sample Command Sequences

What follows are sequences of commands that give examples of how to create motion and logic within a program. All of the commands in this section are buffered-type commands.

Feed to Length

The FL (Feed to Length) command is used for relative or incremental moves. When executed, the motor will move a fixed distance, using linear acceleration and deceleration ramps and a maximum velocity. These move parameters are set using the DI (Distance), AC (Acceleration), DE (Deceleration), and VE (Velocity) commands. The direction of the move is determined by the sign of the DI parameter. "DI80000" is 80000 counts in the CW direction, whereas "DI-80000" is 80000 counts in the CCW direction.



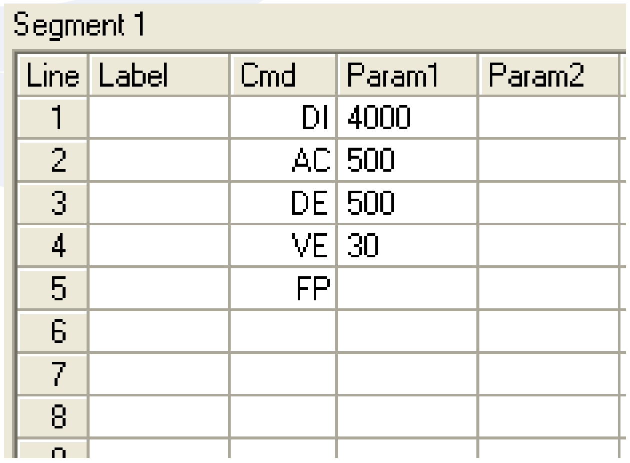
Segment 1

Line	Label	Cmd	Param1	Param2
1		DI	80000	
2		AC	500	
3		DE	500	
4		VE	30	
5		FL		
6				
7				
8				
9				

Here is a sample sequence showing a move of 80000 counts, with a velocity of 30 rps, and accel/decel rates of 500 rps/s. The FL command initiates the move. Also, the order of the commands is not signifcant, except that any changes to the move parameters must be done before the FL command.

Feed to Position

The FP (Feed to Position) command is used for absolute moves. When executed, the motor will move to a position, with linear acceleration and deceleration ramps and a maximum velocity, based on the internal motor position of the drive. The move parameters are set using the AC, DE, VE and DI commands. In the case of the FP command, the DI command sets the motor position, not the relative move distance.



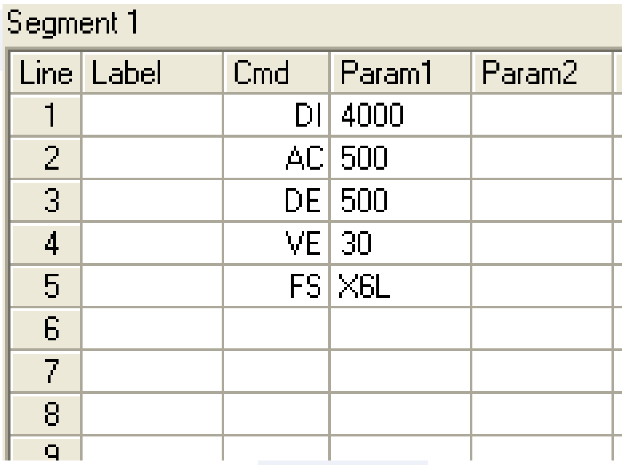
Segment 1

Line	Label	Cmd	Param1	Param2
1		DI	4000	
2		AC	500	
3		DE	500	
4		VE	30	
5		FP		
6				
7				
8				
9				

Here is a sample sequence showing a move to motor position 4000 counts (motor may move CW or CCW depending on the actual motor position before the start of the move), with a velocity of 30 rps and accel/decel rates of 500 rps/s.

Another command to keep in mind when using absolute moves is the SP (Set Position) command. This command allows you to zero the motor position at any time, by entering "SP0", or to set the motor position to another value. The parameter in the SP command is encoder counts. For example with a 2000 line encoder on the motor, an "SP5000" command would set the current motor position to 2.5 revolutions CW from the zero position.

Feed to Sensor



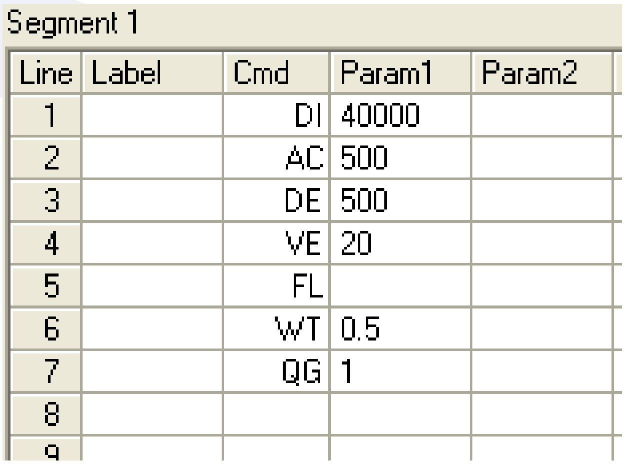
Segment 1

Line	Label	Cmd	Param1	Param2
1		DI	4000	
2		AC	500	
3		DE	500	
4		VE	30	
5		FS	X6L	
6				
7				
8				

The FS (Feed to Sensor) command causes the motor to move at a fixed velocity until an input changes state. When the designated input changes state the motor decelerates to a stop. The parameters of the move are set by the AC, DE, VE and DI commands. In an FS command, the DI command sets both the distance in which the motor should stop after the input changes state and the direction of the move. Parameters for the FS command are the input number (0-7) and the input state the drive should look for: H (high), L (low), R (rising edge), or F (falling edge).

Above is an example where the motor will move in the clockwise direction, starting off with an acceleration rate of 500 rps/s and a maximum speed of 30 rps, until drive input X6 goes low, at which point the drive will use the distance set in the DI command (4000 counts) and the deceleration rate set in the DE command (500 rps/s) to bring the motor to a stop.

Looping



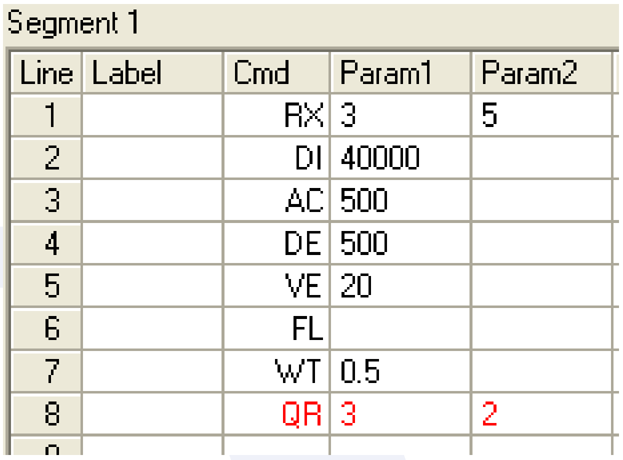
Segment 1

Line	Label	Cmd	Param1	Param2
1		DI	40000	
2		AC	500	
3		DE	500	
4		VE	20	
5		FL		
6		WT	0.5	
7		QG	1	
8				

There are two ways to accomplish looping, or repeat loops, within a program. The first method accomplishes an infinite loop and uses the QG (Queue Goto) command. The parameter for this command is a line number in the segment, and whenever the sequence gets to the QG command the segment will jump to the designated line.

In the example to the right, the sequence contains an FL command, with related parameter commands ahead of it (AC, DE, DI, VE). After the FL command is a WT (Wait Time) command with a time of 0.5 seconds, and then a QG command that points to line 1. This sequence will loop forever now, with the segment always starting at line one after it executes the QG command.

The second method for looping utilizes the QR (Queue Repeat) command. It works by jumping to a given segment line for the number of times indicated in a user-defined data register. Any user-defined data register will work. In the example to the right, the QG command from the previous example has been replaced with the QR command, and parameters have been added. In this sequence the segment will jump to line 2 for the number of times indicated in register 3. Notice on line 1 of the segment that data register 3 has been loaded (using the RX command) with the value 5. Therefore, the FL command in this example (as well as the DI, AC, DE, VE and WT commands) will repeat five times.

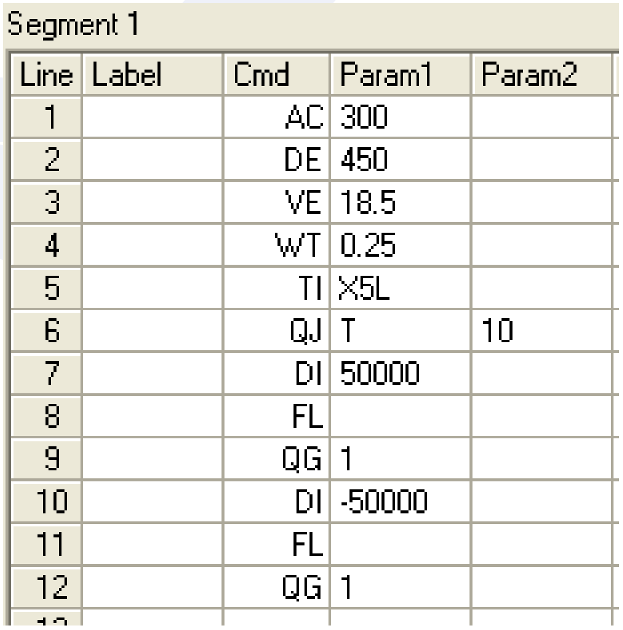

Segment 1

Line	Label	Cmd	Param1	Param2
1		RX	3	5
2		DI	40000	
3		AC	500	
4		DE	500	
5		VE	20	
6		FL		
7		WT	0.5	
8		QR	3	2

Branching

Branching in a program is done using the QJ (Queue Jump) command. Branching is different than looping in that a branch (or jump) is done based on a tested condition. The QJ command will always work in conjunction with one other command: TI (Test Input), TR (Test Register), or CR (Compare Register).

Let's say we have an application with two possible moves. We always want to make a CW move, unless input X5 is low in which case we want to make a CCW move. In this example we set all of the move parameters except distance at the top of the segment. We set accel to 300 rps/s, decel to 450 rps/s, and velocity to 18.5 rps. There is a WT (Wait Time) of 0.25 seconds so that we may have a noticeable delay between moves. Then, we test input X5 to see if it's low using the TI (Test Input) command. If it is true (i.e. input X5 is low), we branch (using QJ) to line 10, set the distance to -50000 counts and make a CCW move. Otherwise the program proceeds to line 7, sets the distance to 50000 counts and makes the CW move. To keep from doing the CCW move right after the CW move, and to repeat


Segment 1

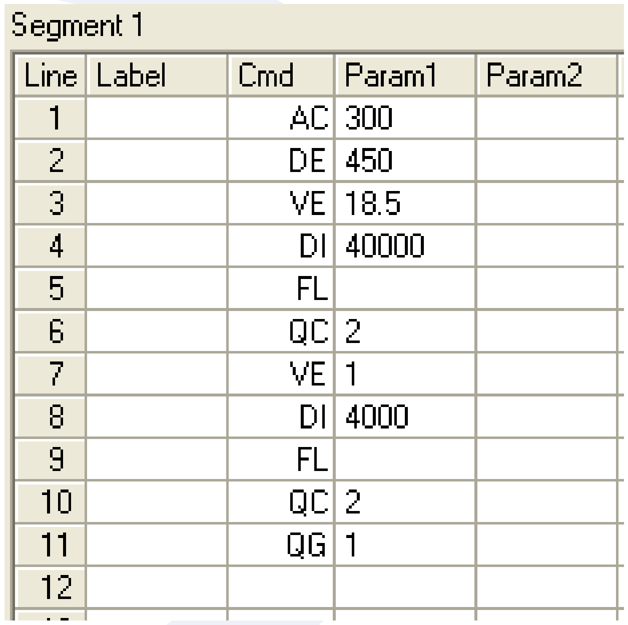
Line	Label	Cmd	Param1	Param2
1		AC	300	
2		DE	450	
3		VE	18.5	
4		WT	0.25	
5		TI	X5L	
6		QJ	T	10
7		DI	50000	
8		FL		
9		QG	1	
10		DI	-50000	
11		FL		
12		QG	1	

the segment forever QG commands are placed after each FL command.

Calling

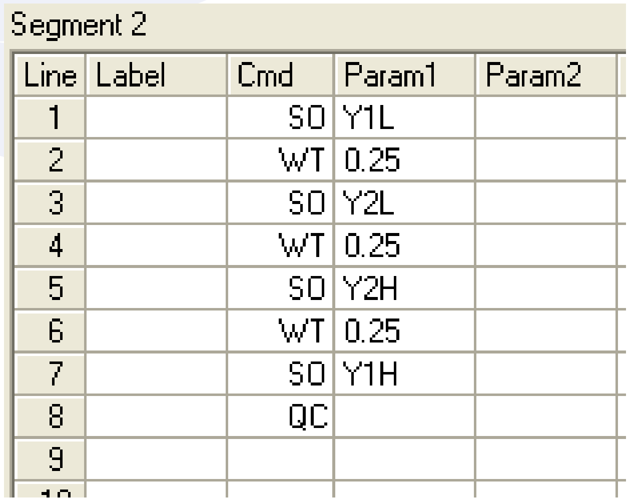
Calling is similar to using sub-routines. The QC (Queue Call) command allows us to exit a segment, execute another segment, and then return to the original segment to the line where the "call" was initiated. This is useful when we have a sequence of commands that is used over and over within a program. Rather than repeatedly program these commands into our segment(s), we locate the frequently-used sequence in its own segment, and then call that segment whenever we need to.

In this example we are making two distinct moves (FL), one fast move and one slow move. After each move we'd like to turn 2 outputs on and off. To accomplish this using the QC command, we must program two segments. In this example, segment 1 is the primary (or calling) segment, and in it we program the two distinct FL commands. We are using the same accel and decel rates for the two moves, but the velocities and distances change. After each move we'd like to set outputs Y1 and Y2 on then off, and rather than entering the necessary commands to do this after each FL command in segment 1, we place the commands in segment 2 and then use the QC command to call it.


Segment 1

Line	Label	Cmd	Param1	Param2
1		AC	300	
2		DE	450	
3		VE	18.5	
4		DI	40000	
5		FL		
6		QC	2	
7		VE	1	
8		DI	4000	
9		FL		
10		QC	2	
11		QG	1	
12				

In segment 2 we place the desired SO (Set Output) commands that turn output Y1 on, then output Y2 on, then output Y2 off and finally output Y1 off. Notice we've also placed WT (Wait Time) commands of 0.25 seconds between each SO command to make the changing output states more noticeable. Segments 1 and 2 work in conjunction when segment 1 reaches its first QC command (with the parameter "2" indicating segment 2). At this moment the program calls segment 2 to execute its sequence of commands. Notice at the end of the sequence in segment 2 we've placed a QC command with no parameter. A QC command with no parameter means return to the original, calling line and segment. So


Segment 2

Line	Label	Cmd	Param1	Param2
1		SO	Y1L	
2		WT	0.25	
3		SO	Y2L	
4		WT	0.25	
5		SO	Y2H	
6		WT	0.25	
7		SO	Y1H	
8		QC		
9				

what happens then is the program returns to segment 1, completes the second move, calls segment 2 again, returns to segment 1 once more, and then starts the process over by looping to line 1 ("QG1").

Multi-tasking

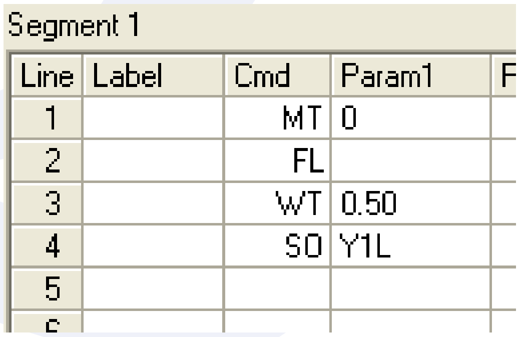
The multi-tasking feature of Q drives allows you to inititate a move command (FL, FP, CJ, FS, etc.) and proceed to execute other commands without waiting for the move command to finish. Without multi-tasking (or more accurately with multi-tasking turned off), a Q drive always executes commands in succession by waiting for the completion of a particular command before moving on to the next command. In the case of move commands, this means waiting for the move to finish before executing subsequent commands. For example, if you have an FL command (Feed to Lenght - incremental move) followed by an SO command (Set Output), the drive will wait to finish the motor move before setting the drive's digital output.

With multi-tasking turned on, a Q drive initiates a move command and then immediately proceeds to execute subsequent commands. For example, doing the same FL and SO commands as above, but this time with multi-tasking turned on, the drive will initiate the move command and immediately proceed to execute the set output command witout waiting for the move command to finish.

Multi-tasking is turned on and off with the MT command. "MT1" turns multi-tasking on, and "MT0" turns it off.

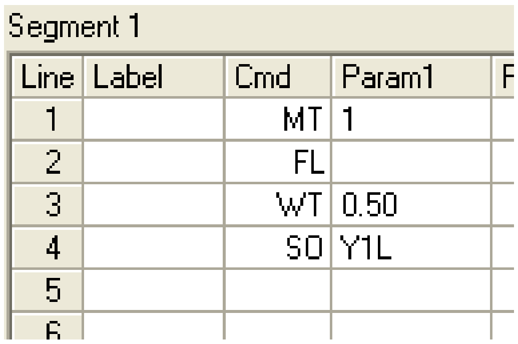
To illustrate the use of the MT command some more, here are a couple of sample command sequences.

In the top command sequence to the right, notice that multi-tasking is turned off, "MT0". When this sequence is exectued by a drive, the FL (Feed to Length) incremental move will complete before the drive waits 0.5 seconds (WT0.50) and then sets output 1 low (SOY1L).


Segment 1

Line	Label	Cmd	Param1
1		MT	0
2		FL	
3		WT	0.50
4		SO	Y1L
5			

In the bottom command sequence to the right, notice that mulit-tasking is turned on, "MT1". When this sequence is executed by the drive, the drive will not wait for the FL command to complete before executing the WT and SO commands. In other words, the drive will initiate the FL command, then wait 0.50 seconds, and then set output 1 low. If the last distance set by the DI command is sufficiently long, the drive's output 1 will be set low before the FL command has completed.


Segment 1

Line	Label	Cmd	Param1
1		MT	1
2		FL	
3		WT	0.50
4		SO	Y1L
5			

This example is actually quite basic, even though it illustrates the function of mult-tasking well. If you try these sequences with your drive, make sure the last DI command is sufficiently large enough to see a noticeable difference in when the drive sets the output.

NOTE: Because it is physically impossible for a motor to make two moves at the same time, move commands are always blocked even with Multi-tasking turned on. For example, if you have Multi-tasking turned on and the program has two move commands in a row, the drive will wait to execute the second move command until the first move command is finished.

MOONS'

Shanghai MOONS'

No. 168 Mingjia Rd. Industrial Park North Minhang District

Shanghai P.R.China 201107

tel / 8621-5263-4688

fax / 8621-5263-4098

www.moons.com.cn

Q User's Guide (Version Ac)
10/29/04